情報処理学会論文誌 Vol.53 No.1 320–335 (Jan. 2012)

センサデータベースマネージャにおける問合せ処理とデータ圧縮の同時最適化

猿渡 俊介[1,a)] 高木 潤一郎[1] 川島 英之[2,3] 倉田 成人[4] 森川 博之[1]

受付日 2011年4月7日, 採録日 2011年10月3日

概要：センサネットワーク技術の進展にともない，多様な実空間情報を取得することが可能になり，データセンタモニタリングや地震モニタリングなどが実現されている．取得したデータを蓄積し，過去のデータを参照することで，現在の状態の把握や傾向の予測などが可能となる．一方で，増え続けるセンサデータを蓄積する際にはデータサイズと問合せ処理性能の劣化が問題となる．本論文の目的は，データベースマネージャにおけるデータサイズ削減と問合せ処理高速化を両立させることである．本論文では，問合せを制限することで性能を向上させる問合せ事前登録方式を基礎として，それに圧縮機能を導入するアプローチを提案する．TIVA データベースマネージャに対して提案アプローチに基づく方式を実装し，データサイズと問合せ処理性能に関する実験評価を行った．その結果，実運用を想定した環境では「結果セグメント圧縮」方式が，データサイズの削減と問合せ処理高速化を最も良く両立させることが分かった．

キーワード：センサデータベースマネージャ，DBM，事前登録方式，再構成，センサネットワーク

Simultaneous Optimization on Query Processing and Data Compression for Sensor Database Managers

Shunsuke Saruwatari[1,a)] Junichiro Takagi[1] Hideyuki Kawashima[2,3]
Narito Kurata[4] Hiroyuki Morikawa[1]

Received: April 7, 2011, Accepted: October 3, 2011

Abstract: Due to the development of sensor network technologies, we can acquire a variety of real-space information, and some applications such as data center monitoring and earthquake monitoring are deployed. Storing sensor data and referring them enable us to get current system status and estimate its trend. However, its huge data size and long latency of query processing are known to be the problems when managing sensor data. The purpose of this paper is to realize both fast query processing and data size reduction. This paper proposes a novel approach to achieve it. The approach is based on a priori query registration method that improves performance by restricting the flexibility of queries for users, and it incorporates a compression function for data size reduction. We implemented multiple methods under the proposed approach on TIVA database manager, and evaluated the methods about data size and query processing. The experimental results showed that "result segment compression" method most appropriately realized both fast query processing and data size reduction in a real operating environment.

Keywords: sensor database manager, DBM, a priori query registration, reorganization, sensor network

1. はじめに

コンピュータの小型化やネットワーク技術の進展，セン

[1] 東京大学先端科学技術研究センター
Research Center for Advanced Science and Technology, The University of Tokyo, Meguro, Tokyo 153–8904, Japan
[2] 筑波大学システム情報系
Faculty of Information, Systems and Engineering, University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan
[3] 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan
[4] 鹿島建設技術研究所
Kajima Technical Research Institute, Kajima Corporation, Chofu, Tokyo 182–0036, Japan
[a)] saru@mlab.t.u-tokyo.ac.jp

サデバイスの充実にともなって，実空間における多様な情報が取得可能となりつつある．携帯電話に具備されたGPSや加速度センサを利用した拡張現実アプリケーションや，物流におけるトレーサビリティサービス，生活空間の情報を記録するライフログサービスなど様々な分野でセンサ情報が利用されている．また，データセンタモニタリング [12] や地震モニタリング [11] に代表されるモニタリングサービスでは，複数のセンサからの情報を収集することで対象の監視を行う．大量に生成されるセンサデータを蓄積し，アプリケーションからの利用に応えるためにはデータ管理用ミドルウェアが必要である．センサデータを蓄積するミドルウェアとして，本論文ではデータベースマネージャ（DBM）[*1] を研究対象として取り上げる．また，本論文ではセンサデータを蓄積するDBMをセンサデータベースマネージャ（SDBM）と表記する．

SDBMは連続的に到着するデータを扱うため，2つの問題に取り組む必要がある．1つ目の問題は，データサイズが大きくなり続けることである．たとえば，PAVENET [17], [18] を用いて100個の加速度センサから100 Hzで10 Byte程度のデータを集める場合，レコード数は1日で10億件，約10 GBとなる．

2つ目の問題は，蓄積データの肥大化にともない，問合せ処理性能が劣化することである．データベース内のデータ増加にともない，インデックスの肥大化による検索コストの増大や，キャッシュヒット率の低下が問合せ処理性能を劣化させる．上記の加速度データを用いて既存のリレーショナルデータベースシステムであるMySQLで検索を行ったところ，検索元のデータ量が1日分のときと地震1回分のときで比較すると処理時間差が40倍程度であった．

1つ目のデータサイズの問題を解決するためには，データをまとめて圧縮する処理が好ましい．圧縮性能は量が多いほど向上するからである．一方，データをまとめた圧縮は問合せ処理の観点からは好ましいことではない．問合せがSDBMに到着するたびに巨大な圧縮データを展開する必要があるからである．圧縮データの展開処理には大きな計算コストとメモリコストが必要になる．

2つ目の問合せ処理性能の問題を解決するために，様々な技術が研究されてきた．問合せ処理性能向上に向けた研究としては，B木などのインデックス，問合せ結果のキャッシング [21]，実体化ビュー [2], [5]，問合せ最適化 [6] などがあげられる．これらの技術は問合せ内容が処理実行前に未知だという前提に立って開発されている．一方，問合せ内容が処理実行前に既知だという前提に立ち，問合せ処理を高速化する技術としてH-Store [10], [19] がある．H-Storeは，問合せに対する結果データをあらかじめ生成してメモリ上に保持することで高速化を実現する．H-Storeは高速であるものの，実行可能な問合せは事前登録したものに限定される．

本論文では，データサイズ削減と高速問合せ処理を両立させるデータ管理手法を提案する．提案手法の基本的アイデアは，H-StoreとSDBMに対する問合せを事前登録させる点にある．事前登録された問合せを解析して，問合せに関連しないデータ領域を一括圧縮することで，データサイズ削減と高速問合せ処理を両立させる．本論文の提案は，データサイズ削減機能を有するH-Storeと学術的に位置付けられる．ここで注意されるべきは，H-Storeには圧縮が考慮されていない点である．H-Storeで提案された問合せ事前登録方式に圧縮機能が導入された技術は，我々が知る限り存在しない．

提案手法の有用性を評価するために，TIVAデータベースマネージャ [22] を開発し，TIVAに提案手法を実装した．そして地震データを用いてTIVA上において実験的に提案手法を評価した．その結果，実運用を想定した我々の実験環境では，提案方式がデータサイズの削減と問合せ処理高速化を同時に最適化することが分かった．

本論文の構成は以下のとおりである．まず2章では提案を述べるための準備として，センサデータベースマネージャについて述べた後，研究目的を示し，本論文の目的に関連する既存研究について述べる．3章では2章で示した研究目的を達成するためのアプローチを提案する．また，提案アプローチを実現するために求められる課題を示し，その課題を解決する手法を提案する．次いで4章で提案手法を実現するSDBMの実装について説明する．さらに5章では実装したSDBMにおけるデータサイズと問合せ処理性能について行った評価実験の結果を示す．最後に6章でまとめと今後の課題を述べる．

2. センサデータベースマネージャ

本研究ではデータベースマネージャ（DBM）を研究対象として取り上げる．DBMはデータベース管理用ミドルウェアの一種である．DBMでは，データをキーと値からなるタプルを原子的単位として扱い，トランザクション処理機能を持たない．本論文ではセンサデータを扱うDBMをSDBMと表記する．

SDBMに対する問合せは，定期監視を目的とする問合せとイベント期間のデータ取得を目的とする問合せの2つに分類される．定期監視を目的とする問合せは，定期的に決められた範囲のデータを取得して解析することで，過去の状態の把握や今後の状態の予測に利用される．たとえば，農業モニタリングでは，前日から過去1カ月分の日照状況のデータを取得することで花弁や農作物の開花時期や生育状況の予測を行う．イベント期間のデータ取得を目的とする問合せは，イベント発生時のイベント前後の範囲でのセンサデータの解析や，複数のイベント間でのデータの比較を行うことで，イベントの詳細な分析を行う．たとえば，

[*1] DBMの例としてはBerkeleyDB [3] やTokyo Cabinet [23] などがある．

地震モニタリングでは，地震発生期間のデータを複数のセンサから取得して，地震波形を比較することで建造物の健全性の分析に利用する．

このように，SDBM に対する問合せはアプリケーション固有の問合せである．また，問合せはアプリケーションに応じてあらかじめ決まるため，事前登録が可能である．さらに，センサデータの時系列性により，各問合せは範囲検索となる．

2.1 研究目的

SDBM の役割は，センサデータを蓄積して参照可能とすることである．センサデータは，X 時 Y 分 Z 秒の値が x というように時刻に関連づけられる値である．加速度センサなら浮動小数値，赤外線センサなら 0/1 などセンサごとに決まった形のデータである．センサネットワークから収集されるセンサデータは，複数のセンサから数 Hz といったサンプリングレートで連続的に発生する．

SDBM では，多数のセンサから連続的に大量のデータが送信されることに起因した 2 つの問題が存在する．1 つ目の問題は，複数のセンサから連続的に到達するセンサデータを蓄積するためにデータサイズが大きくなることである．たとえば，小型センサノード PAVENET [17], [18] を用いて地震モニタリングを行うために加速度センサを超高層ビルに設置することを考える．超高層ビルの揺れを加速度として蓄積して分析することで，その健全性や損傷を評価したり，震災時の避難誘導に役立てたりすることができる．超高層ビルでは，地上の揺れと各階の揺れの関係が重要になる．たとえば，30 階建て超高層ビルの各階 4 隅に加速度センサを設置したとすると，1 棟あたり約 120 個のセンサから 100 Hz で 10 Byte 程度のデータが発生する．センサからのレコード数は 1 日で約 10 億件となり，そのまま蓄積するとデータサイズは約 10.4 GB になる．東京都 23 区内には現在，約 400 棟の超高層ビルがあり，400 棟のビルから得られる年間データサイズは 1.5 PB を超える．これは明らかに削減が求められる量である．

2 つ目の問題は，蓄積されるデータ数の増加にともなって問合せ処理性能が劣化することである．データベース内のデータの増加にともない，インデックスの肥大化による検索コストの増大や，キャッシュヒット率の低下が問合せ処理性能を劣化させる．たとえば，地震モニタリングのための加速度データ 1 日分を既存の RDBMS に登録して地震発生のイベント 1 回分のデータの参照を行ったところ，地震 1 回分のデータのみ登録した場合の約 40 倍の時間を要した．

本研究の目的は，上記 2 つの問題を同時に解決することである．下記ではこれらの問題に対する既存研究について述べる．

2.2 関連研究

2.2.1 データサイズの削減

膨大なデータを扱うデータベース技術として，ウェブ上の膨大なテキストデータを扱うデータベースなど多くの研究が行われている [4]．データを水平分割して複数のサーバに分割して分散ストレージとして並列処理を行うことで増え続けるデータに対応することが可能である [25]．センサデータを扱う場合にも分散化が有効であるが，センサデータはウェブにおけるテキストデータの量をはるかに凌駕する．テキストデータではデータを生成するのは人であるのに対し，センサデータでは機械がデータを生成するからである．ストレージの分散化と同時に，それぞれのストレージでコンパクトにデータを格納することが求められる．

データをコンパクトに格納する方法として，データ圧縮が考えられる [8], [16]．データ圧縮を用いた場合にはデータ圧縮のブロックをどのように扱うかが問題となる．一般的に，データブロックを大きくとった方が圧縮性能は高くなるものの，データ問合せ時にブロックを展開してからでないと必要な部分を取り出すことができないので問合せ処理性能が劣化する．

圧縮技法として，圧縮されたデータから所望の部分のみを展開する技術である Partial Decode [13] が存在する．Partial Decode は，通常の圧縮方式よりも圧縮性能・エンコード性能・デコード性能が劣る．すなわち，Partial Decode は本研究の目的であるデータサイズ削減と問合せ処理高速化には適していない．詳細に関しては，5.4.2 項で議論する．

センサデータに特化した圧縮方式の研究も行われている [7], [15]．具体的には，センサデータの利用目的に対してノイズの除去や特徴量の抽出などを行うことで，前処理とデータサイズの削減を実現している．しかしながら，センサデータの種類や用途が異なるため，特定の不可逆圧縮を適用すると必要な情報が失われる可能性がある．例として，データセンタモニタリングにおける CPU 使用率のデータに対して，サーバを利用するサービスの利用者数の遷移の抽出や，CPU 使用率の突発的な増加などの異常検知を行う場合を考える．利用者数の遷移を抽出したい場合には，低周波成分が必要なためダウンサンプリングなどの手法が有効であるが，異常検知を行うために必要なスパイク部が消えるという問題がある [15]．本論文で研究対象として取り上げる SDBM はミドルウェアである．ミドルウェアは多様な用途に利用されるため，ユーザの意図しないデータ加工は許されない．ミドルウェアおよび SDBM では可逆圧縮の利用が望まれる．以上の理由により，本論文では不可逆圧縮は扱わない．

2.2.2 問合せ処理の高性能化

問合せ処理の高速化を実現するため，数多くの研究が行われてきた．それらには問合せ結果のキャッシング [21],

実体化ビューの作成 [2], [5]，問合せの最適化 [6]，列指向データ格納方式 [9], [20] など多様な手法が含まれる．

上記の高性能化手法は，システムに対する問合せが動的に発行されることを前提として研究開発されてきた．この前提の下では，問合せ内容はシステムに到着するまで不明である．これに対して，問合せがシステムに到着するまで不明であるという前提を崩し，問合せを事前登録することで大きな高性能化を実現する研究がある．H-Store [10], [19] は，問合せをストアドプロジージャとしてあらかじめシステムに登録しておき，問合せに対する結果集合の上位集合を事前生成して保持することで問合せ処理の高速化を実現する．

H-Store が採用している問合せ事前登録方式は，問合せを事前に解析できない従来手法よりも大きな高性能化を実現する．H-Store に基づき開発された VoltDB の技術白書 [24] では，VoltDB が通常の DBMS に対して 45 倍程度の性能であることが示されている．ただし，問合せ事前登録方式では，問合せを事前登録する必要があるために，問合せの自由度が低下する．問合せ事前登録方式は，問合せの自由度を制限することで高性能化を達成しているといえる．

2.2.3 データサイズ削減と問合せ処理高性能化

データサイズ削減と問合せ処理高性能化を同時に扱う研究として，列指向データ格納方式である C-Store があげられる．C-Store ではデータを列単位で扱うデータ構造を持ち，データ構造に合わせた圧縮方式や問合せの実行方式を用いることでデータ圧縮と高速な問合せ処理を実現している [1]．高速性の理由は射影・集約のリレーショナル演算に関するディスク I/O を削減するからである．列指向方式では，行指向方式とは異なり，必要な属性のみディスクからメモリへ転送することが可能となる．

しかしながら，C-Store で高い圧縮性能を達成できるデータ対象は，整列されたデータや同じ値が連続するようなデータなどに限られる．すなわち，2.2.1 項で述べた圧縮技法よりも圧縮性能は劣ると考えられる．なお，C-Store は H-Store [10] や VoltDB 技術白書 [24] で述べられている問合せ事前登録方式を用いていない．

3. 問合せ事前登録方式を利用したデータセグメンテーション

3.1 目的達成へのアプローチと課題定式化

本研究の目的はデータサイズの削減と問合せ処理高速化の両立である．データサイズ削減を実現するために本論文では圧縮技法を用いる．圧縮手法には可逆・非可逆の 2 種類があるが，本論文では可逆圧縮を要するアプリケーションを対象とする．問合せ処理高速化には様々な技法があるが，最も性能が高いと考えられる問合せ事前登録方式を用いる．すなわち，本研究では目的達成のために，問合せ事前登録方式に圧縮機能を追加するアプローチをとる．ここで注意されるべきは，問合せ事前登録方式の既存研究である H-Store では圧縮が考慮されていない点である．H-Store で提案された問合せ事前登録方式に圧縮機能が導入された技術は，我々が知る限り存在しない．

問合せ事前登録方式に圧縮機能を追加する際，データ圧縮セグメントの選定方式が課題となる．圧縮性能の観点から好ましい方式は，セグメントを大きくすることである．しかしながら，セグメントを大きくした場合では，問合せ処理を行う際や，新規データ到着の際に，巨大なセグメントの展開や圧縮が必要になる．データサイズが大きくなればなるほど，展開・圧縮処理により問合せ処理性能が劣化すると考えられる．一方で，セグメントを小さくした場合，展開処理コストが下がるので問合せ処理性能が向上するものの，圧縮性能は劣化すると考えられる．

以上より，圧縮性能と問合せ処理性能を同時に向上させるセグメントの選定は困難であると考えられる．以下では，圧縮性能と問合せ処理性能を同時に向上させるためのセグメントの選定方法を提案する．

3.2 課題解決手法

3.2.1 ナイーブなセグメンテーション手法

最も単純なデータセグメンテーション手法は，一定間隔でデータをセグメンテーションする方式である．一定間隔でセグメンテーションして圧縮した場合の問合せとデータセグメントの関係を**図 1** に示す．一定間隔でセグメンテーションした場合，所定のブロックサイズごとにデータを圧縮する．問合せが参照するデータの範囲と圧縮データブロックの範囲が不一致の場合，余計な圧縮データブロックを展開することになる．余計な展開処理は問合せ処理を劣化させる原因となる．図 1 の場合には，3 つの圧縮データブロックを展開しているが，両端 2 つのデータブロックには問合せ結果と関係がないデータが含まれている．すなわち，図 1 の場合には無駄があり，問合せ処理の性能が劣化する．また，セグメンテーション間隔は変数であるため，適切な間隔を決定する作業が必要になる．データベースが N バイトならば，変数は N 通り存在する[*2]．

3.2.2 提案セグメンテーション手法

3.1 節で述べたように，データサイズを削減するには圧縮が必要である．一方，圧縮をすれば問合せ処理が劣化す

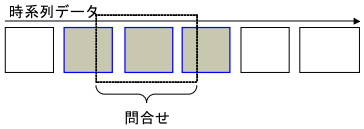

図 1 ナイーブなセグメンテーション手法
Fig. 1 Naive segmentation method.

[*2] ただし，均等分割を考えるならば変数は約 $\sqrt{N}$ 通りである．

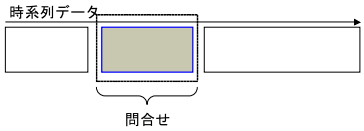

図 2 提案セグメンテーション手法
Fig. 2 Proposed segmentation method.

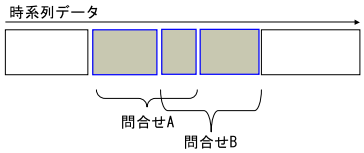

図 3 データベース内セグメンテーション
Fig. 3 Segmentation in a database.

る．本手法はデータサイズ削減と高性能問合せ処理を両立させるために，データ領域を問合せ結果と問合せ結果以外に分割するセグメンテーション方式を提案する．本セグメンテーション方式では，問合せ結果セグメントを圧縮する場合・圧縮しない場合の 2 種類が存在する．これらを各々「結果セグメント圧縮」・「結果セグメント非圧縮」と表記する．圧縮性能については，結果セグメント圧縮方式が明らかに有利である．一方，問合せ処理性能については明らかではない．なぜなら，圧縮には展開処理という欠点があるものの，ディスク I/O コストを低減させるという利点があるからである．

本手法における問合せとデータセグメントの関係を**図 2** に示す．事前登録した問合せに応じて結果データセグメントを設定してデータを圧縮することにより，問合せが参照するデータの範囲と圧縮データブロックの範囲が一致する．問合せ結果となるデータセグメントと展開対象のデータセグメントが一致するため，展開コストを最小化できる．図 1 と図 2 を比較すると，図 1 では無駄な展開処理が求められる一方，図 2 では無駄な展開処理がない．したがって，提案手法はナイーブな手法よりも問合せ処理性能面で優位だと考えられる．

図 3 に，提案手法を用いた場合のデータベース内のセグメント例を示す．図 3 では，問合せ A の範囲検索と問合せ B の範囲検索が事前登録されている．連続的に並ぶセンサデータに対して各問合せの範囲検索に合わせてセグメンテーションを行う．また，問合せ A と問合せ B のように，問合せ結果となるデータの範囲が重複する部分は別のセグメントとして管理される．

図 4 に問合せの検索範囲が包含関係になった場合のセグメンテーションを示す．図 4 では，時間範囲 $start_A$〜end_A を持つ問合せ A と，時間範囲 $start_B$〜end_B を持つ問合せ B があり，問合せ B が問合せ A を包含している．この場合，提案手法では $start_B$〜$start_A$，$start_A$〜end_A，end_A〜end_B の 3 つのセグメントを生成する．

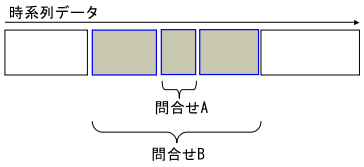

図 4 問合せに包含関係がある場合のセグメンテーション
Fig. 4 Segmentation which has inclusion relation.

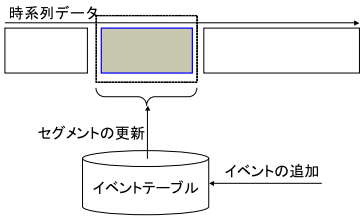

図 5 イベントデータに応じたセグメントの更新
Fig. 5 Segmentation update according to event data.

3.3 再編成処理

SDBM は連続的にセンサデータを受信するため，時間の経過とともに問合せ結果が変化する．したがって，データセグメントの再編成を行う必要がある．SDBM では，センサデータは追記のみで更新や削除が行われないため，セグメントの区切りを変えることだけで再編成が実現できる．再編成のタイミングは登録された問合せの特性によって異なる．たとえば，「前日から過去 1 カ月」の範囲検索では，1 日経過することをトリガとしてデータ範囲を再度算出し，セグメントのオンライン再編成を行う．

イベント発生期間のデータ取得を目的とする問合せはイベントテーブルと連携することで実現される．イベントテーブルとは，イベント情報をイベントのメタ情報，開始時刻，終了時刻を組にして管理しているデータベースである．**図 5** にイベントテーブルによるセグメントの更新処理を示す．イベントテーブルに対して外部からイベント情報が入力されると，イベント情報が持つ開始時刻，終了時刻を用いてセグメンテーションを行って問合せ結果データセグメントを生成する．たとえば，地震モニタリングでは地震が発生したという情報がイベント情報としてイベントテーブルに登録され，地震波を含む加速度データが問合せ結果データセグメントとなる．

4. SDBM システム TIVA の実装

3 章に示したデータ管理手法を TIVA データベースマネージャ（DBM）[22] に実装した．TIVA の実装は Linux 上で C 言語を用いてライブラリとして行った．コードサイズは 3,500 行程度である．**図 6** に実装したシステムの全体像を示す．本システムは，初期化機構，データ挿入処理機構，問合せ登録機構，問合せ処理機構，再編成機構，イベント管理機構の 6 つの機構から構成される．それぞれの機構はメモリやディスク上に配置されたセンサデータ群，問

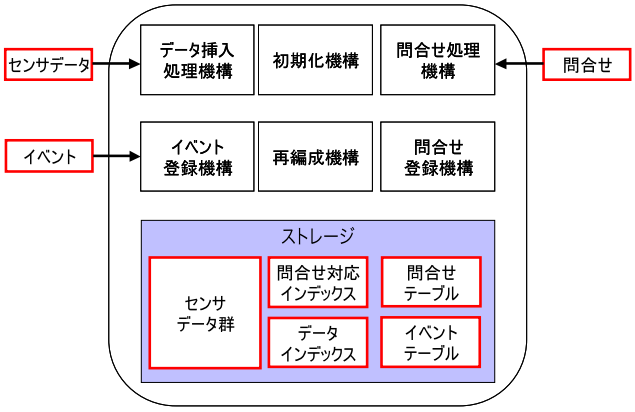

図 6 システム全体像
Fig. 6 System overview.

合せ対応インデックス，問合せテーブル，データインデックス，イベントテーブルを利用してデータ管理を行う．問合せテーブルとイベントテーブルの実装には MySQL-5.5.9 を用いた．

4.1 初期化機構

初期化機構はデータベースを初期化して必要な要素を作成する機構である．ユーザプログラムに対して `tiva_db_create` 関数と `tiva_etable_create` 関数を提供する．

`tiva_db_create` 関数はデータベースを作成する関数である．倍精度浮動小数点数で表現される X，Y，Z 軸の加速度を蓄積するデータベースを作成する場合には

`tiva_db_create("eq", TYPE_DOUBLE, 3)`

のようにデータベースの名前，センサ値の型，1 レコードあたりのセンサ値の個数を引数として呼び出す．`tiva_db_create` 関数が呼び出されるとセンサデータ群，問合せ対応インデックス，問合せテーブル，データインデックスが生成される．

`tiva_etable_create` 関数はイベントテーブルを作成するための関数である．イベントテーブルは地震などのイベント情報とイベントに対応するセンサ値を結びつける．たとえば，震度情報を具備する地震イベントを蓄積するためのテーブルは

`tiva_etable_create("eq", "eqscale as int")`

のようにイベントテーブルを関連付けるデータベースと，イベント情報が具備するメタデータの情報を引数として呼び出す．

4.2 データ挿入処理機構

データ挿入処理機構はセンサからのデータをデータベースに格納する機構である．ユーザプログラムに対して `tiva_add` 関数を提供する．センサからのデータは，3 軸加速度データの場合，

`tiva_add(`*time*, *x*, *y*, *z*`)`

のように時刻と 3 軸の加速度データを引数として呼び出す．

`tiva_add` 関数が呼ばれると，まず，追記専用バッファに対してデータを追記する．バッファにデータが一定量たまると，センサデータ群のファイルに対して追記する．このとき，データインデックスにアクセスしてインデックスの更新を行う．センサデータは時刻に関連付けられることから，本システムでは時刻をキーとするインデックス付けを行う．また，センサデータが時系列にあらかじめ並んでいることを利用して，軽量なインデックス方式である Sparse Index [14] を用いた．

4.3 問合せ登録機構

問合せ登録機構は，問合せを事前に登録するための機構である．ユーザプログラムに対して `tiva_qadd` 関数を提供する．`tiva_qadd` 関数の引数には問合せを表す文字列が渡される．たとえば，加速度センサのデータを蓄積したデータベースに対して，ある時間帯の震度 3 以上の地震の波形データの取得を行う問合せを事前登録する場合，

`tiva_qadd("tiva_get(tiva_event('eqscale>=3'),`
`tiva_time(`*start*, *end*`))")`

のように記述する．`tiva_get` 関数に関しては後述する．

`tiva_qadd` 関数が呼ばれると，引数に渡された問合せのハッシュ値を計算する．ハッシュ値と，登録する問合せの引数を問合せテーブルに追加する．問合せテーブルへの追加が終了すると再編成機構の `tiva_reorganize` 関数を呼び出して再編成を行う．

4.4 問合せ処理機構

問合せ処理機構は，ユーザプログラムからの問合せに応じて結果を返す機構である．ユーザプログラムに対して `tiva_get` 関数を提供する．`tiva_get` 関数の引数にはセンサの位置情報などのメタ情報，地震の発生などのイベント情報，時間範囲を指定する時刻情報が含まれる．たとえば，加速度センサのデータを蓄積したデータベースから，ある時刻 *start* から時刻 *end* までの時間帯の震度 3 以上の地震波形データの取得を行う問合せは，

`tiva_get(tiva_event('eqscale>=3'),`
`tiva_time(`*start*, *end*`))`

のように記述する．`tiva_event` 関数は引数にイベント情報のメタ情報を指定する構造体を返す関数である．`tiva_time` 関数は検索時間範囲を指定する構造体を返す関数である．

`tiva_get` 関数が呼ばれると，問合せ処理機構は問合せテーブルを参照し，問合せがすでに登録されているかどうか判定を行う．問合せが事前に登録されたものである場合，問合せインデックスから対象となるセンサデータのインデックスを取得する．取得したインデックスを用いて，データセグメントを参照し，ユーザプログラムに対してセンサデータを返す．問合せが登録されていなかった場合，

イベントテーブルを参照して，検索対象となっているイベントの開始時間と終了時間を取得する．取得した開始時間と終了時間を用いてデータインデックスを参照する．データインデックスの情報より必要なデータが含まれるデータセグメントを決定し，データセグメントを展開する．展開したデータの中から問合せに一致する部分のみを抜き出してユーザプログラムに対してセンサデータを返す．

4.5 再編成機構

再編成機構は，データセグメンテーションとセグメントの圧縮を行う機構である．ユーザプログラムに対して `tiva_reorganize` 関数を提供する．ユーザプログラムは定期的に `tiva_reorganize` 関数を呼び出す．たとえば，地震モニタリングでは `cron` によって 1 時間に 1 回呼び出す．ユーザプログラムからの呼び出しに加えて，`tiva_reorganize` 関数は，問合せ登録機構に新しい問合せが登録されたときと，イベントテーブルに新しいイベントが登録されたときにも呼び出される．

`tiva_reorganize` 関数が呼び出されると，問合せテーブルに登録されている全問合せを参照する．各問合せの問合せ条件を取得し，問合せ条件を展開して問合せの対象となっているセンサデータの範囲セットを取得する．取得した範囲セットを用いて，データインデックスとセンサデータ群にアクセスしてセグメンテーションを行う．生成された各セグメントに対して圧縮を行い，適用した圧縮形式をデータインデックスに登録する．それと同時に，問合せインデックスにアクセスして問合せとセグメントの関連付けを行う．

4.6 イベント管理機構

イベント管理機構はイベントデータを管理するための機構である．ユーザプログラムに対して `tiva_add_event` 関数を提供する．`tiva_add_event` 関数の引数はアプリケーション依存である．たとえば，時刻 *start* から時刻 *end* の間に発生した震度 3 の地震イベントを追加する場合には

```
tiva_add_event("eqscale=3",
                tiva_time(start, end))
```

のように記述する．`tiva_add_event` 関数が呼ばれるとイベントテーブルにイベント情報が追加され，再編成機構の `tiva_reorganize` 関数を呼び出して再編成を行う．

5. 評価

5.1 実験環境

本章では，4 章に示した TIVA を用いて，3 章に示したデータ管理手法の評価を実験により行う．実験のために，実運用計算機と詳細評価用計算機の 2 種類を用意した．

実運用計算機では，センサデータを蓄積する際に実際に用いる環境を想定しており，実運用環境での性能を検証する．OS が Ubuntu 8.04 LTS，CPU が Intel Xeon X3210 2.13 GHz，メモリが ECC2 DDR2 SDRAM 8 GB，ハードディスクは容量が 2 TB の RAID 1+0 である．本実験で用いた RAID は，HP の Smart アレイ E200（128 MB バッテリバックアップ式ライトキャッシュ）とハードディスク（1 TB，7,200 RPM，32 MB キャッシュ）4 台で構築されている．

詳細評価用計算機では，ストレージの交換や CPU 性能の変更が容易であるものを用いており，ハードウェア性能などのパラメータを変更した場合の性能の変化を検証する．OS が Ubuntu 10.10，CPU が CPU コアを 2 つ搭載した Intel Core i5 520M 2.40 GHz，メモリが DDR3 SDRAM 4 GB，ストレージはハードディスクとソリッドステートドライブの 2 種類である．詳細評価用計算機のハードディスクは TOSHIBA の MK645GSX（640 GB，5,400 RPM，8 MB キャッシュ，平均シーク時間 12 ms），ソリッドステートドライブは Transcend TS120GSSD25D-M（120 GB，64 MB キャッシュ）である．

実験では，TIVA にセンサデータを登録し，再編成後のデータサイズと問合せ処理性能の評価を行った．実験では PAVENET [17], [18] を用いて地震の計測を行っている 3 軸加速度センサから 100 Hz で収集した 1 億件の加速度データを利用した．加速度データは $(time, x, y, z)$ のように時刻とセンサ値のペアからなる値であり，時刻は `struct timeval` 型，センサデータはそれぞれ倍精度浮動小数型で表されるため，1 レコードのデータサイズは 40 バイトである．圧縮アルゴリズムには LZO（Lempel-Ziv-Oberhumer）を利用した．また，各評価では互いに重複しない 10 個の検索クエリを用い，各検索クエリの検索時間範囲のサイズを 0.1 秒から 10 万秒まで変化させた．

問合せ処理性能とデータの縮小度を相対的に評価するため，次の 4 手法を用いた．

(1) 圧縮なし（Non-compression）
この手法はデータベースの圧縮を行わない手法である．他手法の性能を測るために求められるベースラインとなる．圧縮をしないためデータサイズが大きくなることが予想される．また，問合せを事前登録しない方式であるため，問合せ処理性能が劣ることも予想される．

(2) 一様な圧縮（Uniform Compression）
この手法はデータベースを一定間隔ごとにセグメンテーションし，セグメントごとに圧縮する手法である．また，問合せを事前登録しない方式であるため，問合せ処理性能が劣ることも予想される．なお，以下の 2 手法はすべて問合せを事前登録する方式であることに注意されたい．

(3) 結果セグメント圧縮（Result Segment Compression）
この手法は 3.2.2 項で提案した手法である．2 種類の

圧縮セグメントを作成する．1 つ目の圧縮セグメントは，事前に登録した問合せの結果集合以外のデータ領域であり，「一様な圧縮」と同様に一定間隔ごとにセグメンテーション・圧縮される．2 つ目の圧縮セグメントは，事前に登録した問合せの結果集合であり，各結果セグメントごとに圧縮される．

(4) 結果セグメント非圧縮（Result Segment Non-compression）

この手法は 3.2.2 項に示した手法である．2 種類のセグメントを作成する．1 つ目のセグメントは，事前に登録した問合せの結果集合以外のデータ領域であり，「一様な圧縮」と同様に一定間隔ごとにセグメンテーション・圧縮される．2 つ目のセグメントは，事前に登録した問合せの結果集合であり，圧縮されない．

本研究におけるベースラインは問合せを事前登録しない「圧縮なし」であり，H-Store [10] の方式とは異なる点に注意されたい．H-Store 方式をベースラインとしなかった理由は以下の 2 点である．

(1) H-Store 方式はオンメモリで動作すること

H-Store はすべてのデータをオンメモリに置くことを前提とする．しかしながら，本研究では，データは二次記憶に置くことを前提とする．SDBM においてはメモリに置ききれない程度の大規模データを扱うからである．

(2) 研究対象が DBM であること

理由 (1) に加えて，本研究の対象が DBMS（データベース管理システム）ではなく，データベースマネージャ（DBM）である点も H-Store 方式をベースラインにしなかった理由である．H-Store は DBMS を前提とした方式である点に留意されたい．DBM における問合せにおいて，事前登録方式は我々の知る限り存在しない．それゆえ，ベースラインは問合せを事前登録しない圧縮なし手法にすることが，DBM における研究として見たときには適切であると考えられる．

また，「一様な圧縮（Uniform Compression）」では，検索時間範囲が 100 秒（1 万タプルに相当）のときに最適になるようにセグメントサイズを 10 万タプルとした．提案手法の応用の 1 つとして視野に入れている地震モニタリングでは，地震に対する検索が多くなされる．地震は長いもので数分であり，100 Hz でサンプリングした場合には数万タプルに相当する．「一様な圧縮」において，問合せ結果数が 1 万タプルの際のセグメントサイズと問合せ処理時間を**図 7** に示す．実験では実運用計算機を用いた．図 7 より，問合せ結果数が 1 万タプルの際にはセグメントサイズは 10 万タプルが最適となることが分かる．

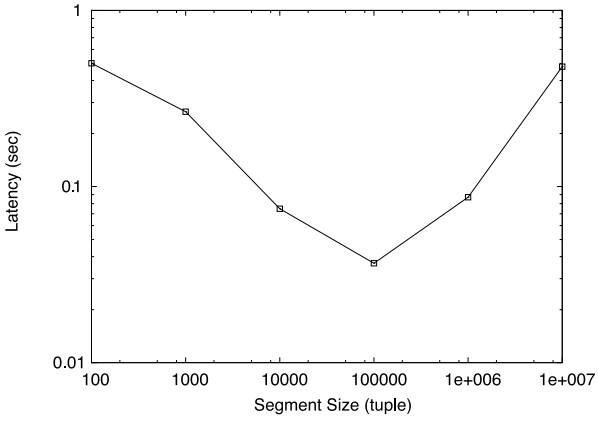

図 7 一様な圧縮におけるセグメントサイズと問合せ処理時間
Fig. 7 Segment size vs. latency in uniform compression.

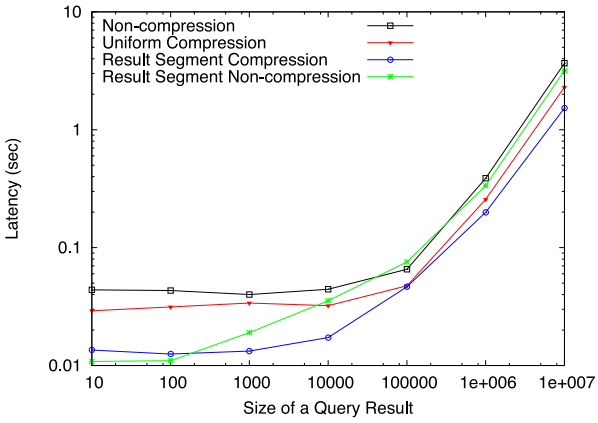

図 8 問合せ結果数に対する問合せ処理時間
Fig. 8 Size of a query result vs. latency.

5.2 問合せ処理性能に関する評価

5.2.1 結果

5.1 節で述べた実運用計算機を用いた場合の問合せ処理時間に関する結果を**図 8** に示す．評価では，1 つの問合せのみが同時に実行される際の時間の計測を行った．また，登録されている検索クエリは互いに重複しないものとした．図の横軸と縦軸は各々問合せ結果数と，問合せ処理時間である．図 8 からは以下の 3 つの注目されるべき結果が観察される．

1 つ目の結果は，「結果セグメント圧縮（Result Segment Compression）」がほぼ最良であることである．問合せ結果数（横軸）が 1,000 未満の場合には「結果セグメント非圧縮（Result Segment Non-compression）」が最良であるが，それ以上の場合には「結果セグメント圧縮」が最良である．

2 つ目の結果は，「結果セグメント非圧縮」の性能が問合せ結果数（横軸）が 100,000 以上の場合に「圧縮なし（Non-compression）」とほぼ同様である点である．また，このときの性能は最悪である．

3 つ目の結果は，問合せ結果数（横軸）が 100,000 の場合に，「一様な圧縮」と「結果セグメント圧縮」がほぼ等しいことである．それ以外の場合には「一様な圧縮」は「結

果セグメント圧縮」よりも悪くなっている．

5.2.2 考察

1つ目の結果において「結果セグメント圧縮」がほぼ最良であるのは，セグメントサイズが最適であるからだと考えられる．ただし，問合せ結果数（横軸）が1,000未満の場合には展開処理コストが大きくなるために，結果セグメントを圧縮せずに保持する「結果セグメント非圧縮」が最良になっていると考えられる．

2つ目の結果において「結果セグメント非圧縮」が「圧縮なし」とほぼ同一になった理由は，ディスクI/Oコストが原因だと考えられる．問合せ結果数（横軸）が小さいときに「結果セグメント非圧縮」が最良である理由は上述した．逆に問合せ結果数（横軸）が大きくなると，巨大な問合せ結果をディスクからメモリへ転送するコストが大きくなり，性能が劣化すると考えられる．

3つ目の結果において「一様な圧縮」と「結果セグメント圧縮」が等しく最良になる理由は，両者のセグメントサイズが一致したからだと考えられる．今回の実験では，5.1節で述べたように，「一様な圧縮」のセグメントサイズを10万件に設定した．問合せ結果数（横軸）が10万件のとき，「一様な圧縮」の挙動は「結果セグメント圧縮」と同一になる．したがって，実験結果がほぼ等しくなったと考えられる．「一様な圧縮」において，問合せ結果数がセグメントよりも小さい場合には無駄な展開処理が発生するために「結果セグメント圧縮」よりも悪くなる．また，「一様な圧縮」において，問合せ結果数がセグメントよりも大きい場合には，セグメントの読み込みが複数回発生することによるシーク時間や回転待ち時間のオーバヘッドが重畳されることで「結果セグメント圧縮」よりも悪くなると考えられる．

以上より，セグメントサイズと問合せ処理性能には密接な関係があると考えられる．

5.3 データベースサイズ縮小化に関する評価

5.3.1 結果

データベースサイズに関する結果を**図9**に示す．圧縮を行わない場合，1億レコードのデータベースサイズは4 GBとなる．これは「圧縮なし」の結果に相当する．図9から，次の注目されるべき3つの結果が観察される．

1つ目の結果は，「結果セグメント圧縮」，「結果セグメント非圧縮」，「一様な圧縮」の3手法は問合せ結果数（横軸）が10件から10万件までの間は，ほぼ等しく最良であることである．一方で，「圧縮なし」はつねに最悪の結果を示している．

2つ目の結果は，問合せ結果数（横軸）が100万件以上になると，「結果セグメント非圧縮」の性能が急速に劣化することである．問合せ結果数（横軸）が1,000万件のとき，「結果セグメント非圧縮」と「圧縮なし」は同じ結果になっている．

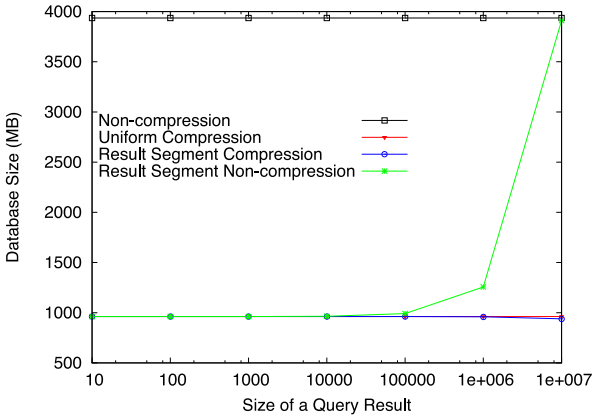

図9 データベース全体のサイズ比較
Fig. 9 Size of a query result vs. database size.

表1 データベース全体のサイズ比較
Table 1 Size of a query result vs. database size.

手法	問合せ結果数		
	10万件	100万件	1,000万件
一様な圧縮	961.748 MB	961.748 MB	961.748 MB
結果セグメント圧縮	961.576 MB	959.596 MB	939.124 MB

3つ目の結果は，問合せ結果数（横軸）が10万件以上になると，「結果セグメント圧縮」のデータベースサイズが「一様な圧縮」よりもわずかに小さくなることである．問合せ結果数（横軸）が10万件以上の結果を**表1**に示す．表1より「結果セグメント圧縮」は「一様な圧縮」よりも最大で2.36%程度データベースサイズを削減することが分かる．

5.3.2 考察

1つ目と2つ目の結果の理由は明らかである．圧縮がデータベースサイズ削減に大きな寄与をしたと考えられる．

3つ目の結果において，「結果セグメント圧縮」が「一様な圧縮」よりも最大で2.36%程度データベースサイズを削減した理由は，圧縮対象データのサイズと圧縮性能に相関があるからだと考えられる．上記実験で用いたLZOは辞書式アルゴリズムである．圧縮対象データのサイズが大きいほど，優れた辞書を作成可能である．すなわち，問合せ結果数が増えるほど，優れた辞書が作成され，圧縮性能が向上したのだと考えられる．

表1において問合せ結果数が100倍違うにもかかわらず，圧縮性能の向上が2.36%程度にとどまったのは圧縮するデータサイズがある一定以上大きくなると圧縮性能の向上度合いが小さくなることに起因する．**図10**にデータサイズと圧縮率の関係を示す．横軸がデータサイズ，縦軸が圧縮率である．圧縮方式であるLZO，ZLIB，LZMAの圧縮率の変化を描画している．図10から分かるように，データサイズが小さいうちには圧縮性能の向上が大きいが，データサイズが大きくなると圧縮性能の向上が小さくなる．

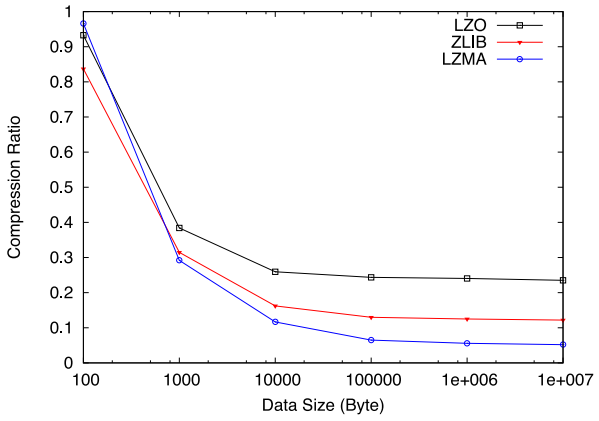

図 10 データサイズに対する圧縮率の変化
Fig. 10 Data size vs. compression ratio.

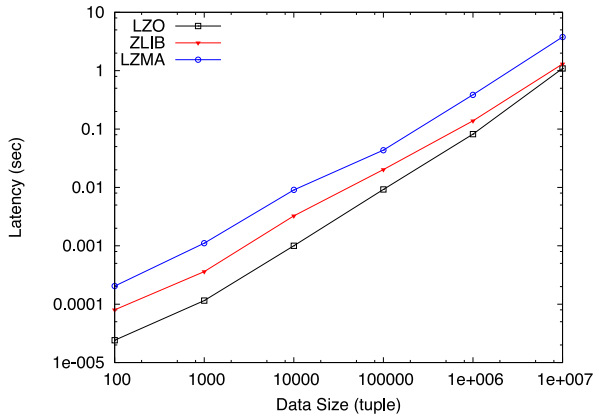

図 11 圧縮方式の違いによる展開処理時間
Fig. 11 Data size vs. latency under different compression algorithm.

5.4 結果セグメント圧縮方式に関する議論

5.4.1 展開処理性能と圧縮性能に関する議論

5.2 節と 5.3 節から，提案手法の「結果セグメント圧縮」が高性能問合せ処理とデータサイズ縮小化を同時に最適化する手法であることが示された．ただし，問合せ結果数が小さいときには「結果セグメント圧縮」は「結果セグメント非圧縮」よりも問合せ処理性能が若干劣る．この劣化の理由は展開処理コストであると考察した．もしも展開処理を高速化できれば，「結果セグメント圧縮」をより高性能化することができる．5.2 節と 5.3 節で用いた圧縮アルゴリズムは LZO であったが，本節では，圧縮アルゴリズムと「結果セグメント圧縮」との性能の関係を調べる．

調査では圧縮方式として，LZO，ZLIB，LZMA の比較を行った．まず，5.3.2 項に示した図 10 から，LZMA や ZLIB では LZO に比べてさらに高い圧縮性能を実現していることが分かる．

次に，**図 11** にデータサイズを変えた場合の各圧縮アルゴリズムの展開処理時間を示す．処理時間の計測は，5.1 節で述べた詳細評価用計算機において，ソリッドステートドライブを用いて行った．横軸がデータサイズ，縦軸が圧縮データの展開に要した時間である．図 11 より，高い圧縮性能を実現する LZMA の展開時間が最も長く，圧縮性能の低い LZO の展開時間が最も短いことが分かる．

上記より，圧縮性能の高いアルゴリズムほど展開処理が長くなることが示された．圧縮アルゴリズムを変えるだけでは，「結果セグメント圧縮」方式において問合せ処理性能を改善できないことが分かった．

5.4.2 部分展開可能な圧縮法に関する議論

5.4.1 項で述べた圧縮方式は圧縮データの一部分のみを取り出して展開することはできない．それに対し，圧縮技法として，圧縮されたデータから所望の部分のみを展開する技術である Partial Decode [13] が存在する．本項では Partial Decode の利用可能性について議論する．

まず，データサイズ削減に関しては，文献 [13] から分かるように，Partial Decode を用いた場合には圧縮性能が既存の圧縮方式よりも劣化する．

次に，問合せ処理性能に関しても，Partial Decode の性能は提案手法よりも劣化する．データベースから検索結果を取得する場合では，ストレージからの圧縮データの読み出しと圧縮データの展開処理の 2 つをともなう．Partial Decode は部分展開を可能とするが，圧縮性能・エンコード性能・デコード性能が bzip や gzip よりも悪くなる．すなわち，I/O コストは既存の圧縮方式よりも劣化する．また，デコードすべきデータ量が増大するうえに，単位バイトあたりのデコード性能が劣化するため，圧縮データの展開処理に関する性能も低下する．

ただし，ここでの議論では，Partial Decode を用いた場合には部分展開を可能とするために圧縮性能・展開速度が既存の圧縮方式に比べて悪くなることを前提としている点に注意されたい．仮に，部分展開が可能であり，かつ圧縮性能・展開速度が既存の圧縮方式よりも優れる Partial Decode が今後実現されるならば，Partial Decode を提案手法と組み合わせることを積極的に考える必要があろう．

5.5 ハードウェア性能に関する議論

5.2 節で述べた実運用計算機の評価では，「結果セグメント圧縮」が最適となるとの結果が得られた．しかしながら，問合せ処理性能はディスクの読み出し速度，CPU 性能に依存する圧縮データの展開速度などハードウェアのスペックによって異なると考えられる．本節では，ハードウェアスペックとの関係に関して，提案方式である「結果セグメント圧縮」と，「結果セグメント非圧縮」とを比較しながら議論する．

まず，「結果セグメント圧縮」の処理速度 R_c（Byte/s）について考える．ある任意のストレージの読み出し速度を r（Byte/s），ある任意の圧縮方式の展開速度を s（Byte/s），圧縮率を p とおいたとき，サイズ D（Byte）の問合せ結果を取り出すのに必要な時間を t（s）とする．ここで圧縮率

p は，圧縮後のデータサイズを圧縮前のデータサイズで除算して算出される $p > 0$ の値をとり，p が 0 に近ければ近いほど圧縮性能が高いことを意味する．t（s）はストレージから圧縮された問合せ結果を読み出す時間と，圧縮データを展開する時間で構成される．ここで，読み出し時間と圧縮データの展開時間以外の，クエリの展開，セグメントのオープン・クローズなどの処理時間が無視できるほど問合せ結果数 D が十分に大きいとする．ストレージから圧縮された結果を読み出す時間は，圧縮率 p で圧縮された D（Byte）をストレージから読み出す時間となるため $\frac{Dp}{r}$ と表すことができる．同様に，問合せ結果を展開する時間は $\frac{Dp}{s}$ と表すことができる．すなわち，全体の問合せ処理時間 t は

$$t = \frac{Dp}{r} + \frac{Dp}{s} \tag{1}$$

となる．D を D バイトの問合せ結果を取り出すのにかかった時間 t で除算するとストレージからの読み込みと展開を合わせた全体の処理速度 R_c（Byte/s）が得られる．

$$\begin{aligned} R_c &= \frac{D}{t} \\ &= \frac{D}{\frac{Dp}{r} + \frac{Dp}{s}} \\ &= \frac{1}{\frac{p}{r} + \frac{p}{s}} \\ &= \frac{rs}{sp + rp} \end{aligned} \tag{2}$$

次に「結果セグメント非圧縮」の処理速度 R_{nc} について考える．問合せ結果数 D が十分に大きい場合，「結果セグメント非圧縮」の問合せ結果を取り出す速度 R_{nc} はストレージの読み出し速度と一致するため，

$$R_{nc} = r \tag{3}$$

となる．

例として，**図 12** に，式 (4) において圧縮率 p が 0.25，ストレージの読み出し速度 r を変化させたときの「結果セグメント圧縮」と「結果セグメント非圧縮」の実効処理速度の比較を示す．横軸がストレージの読み出し速度 r，縦軸が問合せ処理速度 R_c および R_{nc} である．「結果セグメント圧縮」の処理速度 R_c として，CPU の性能が異なる場合を模して圧縮データの展開速度 s が 10 MB/s，100 MB/s，1,000 MB/s の 3 種類を描画している．黒色の直線は「結果セグメント非圧縮」を表す．

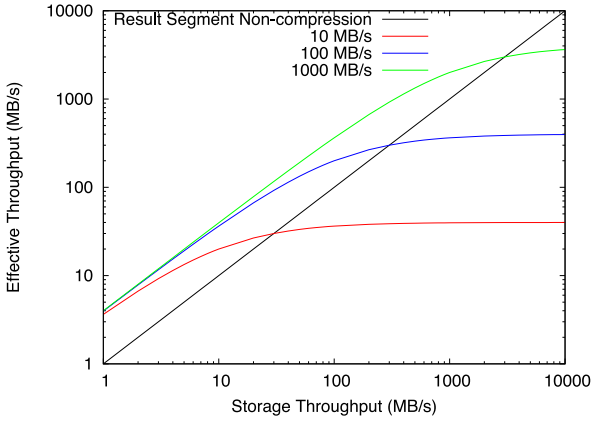

図 12 ストレージ性能と CPU 性能が変化した場合の処理速度
Fig. 12 Storage throughput vs. effective throughput under different CPU performance.

図 12 より明らかなことは，ストレージのスループットが低いほど「結果セグメント圧縮」が有利であることと，ストレージのスループットが高いほど「結果セグメント圧縮」が不利であることである．すなわち，CPU の性能とストレージ性能によって「結果セグメント圧縮」と「結果セグメント非圧縮」の優劣が決まる．

「結果セグメント圧縮」の問合せ処理速度 R_c が「結果セグメント非圧縮」の問合せ処理速度 R_{nc} よりも大きくなる条件

$$R_c > R_{nc}$$

に対して，式 (2) と式 (3) を代入して解くと，

$$(1 - p)s > pr \tag{4}$$

が得られる．すなわち，式 (4) を満たすときに「結果セグメント圧縮」が最適となる．

式 (4) の妥当性を検証するために，5.1 節で述べた詳細評価用計算機を用いて実測を行った．圧縮方式には圧縮性能が低くて展開速度の大きい LZO と，圧縮性能が高くて展開速度の小さい LZMA の 2 種類を用いた．事前実験では，ハードディスクの読み出し速度は最大で約 85.9 MB/s，ソリッドステートドライブの読み出し速度は最大で約 269 MB/s，LZO の圧縮率は最小で約 0.289，LZO の展開速度は最大で約 115 MB/s，LZMA の圧縮率は最小で約 0.0570，LZMA の展開速度は最大で約 6.33 MB/s であった．

ストレージとしてハードディスク，圧縮方式として LZO を用いた場合の問合せ処理時間を**図 13** に示す．横軸が問合せ結果数，縦軸が問合せ処理時間である．図 13 では，「結果セグメント圧縮」の総処理時間（total），ディスクからの読み出し時間（read），圧縮データの展開時間（decode），「結果セグメント非圧縮」の総処理時間（result segment non-compression）の 4 つを描画している．図 13 から分かるように，検索サイズによらずつねに「結果データ圧縮」の処理時間が「結果セグメント非圧縮」に比べて優位であった．また，ハードディスクと LZO の実測データを式 (4) に代入すると

$$81.8 \times 10^6 > 24.8 \times 10^6$$

が得られ，式 (4) が成り立つ．すなわち，「結果セグメント圧縮」が「結果セグメント非圧縮」よりも処理速度が大き

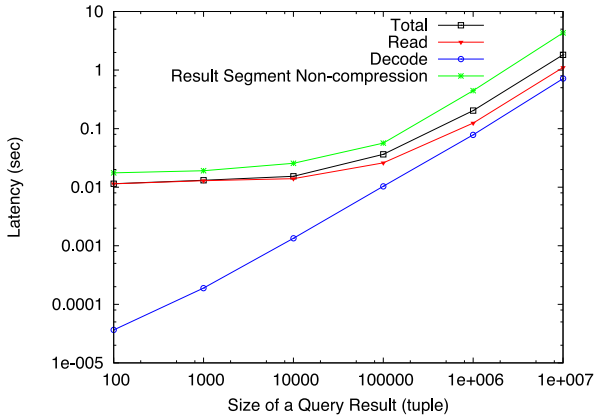

図 13 問合せ結果数に対する問合せ処理時間（ハードディスクと LZO）
Fig. 13 Size of a query result vs. latency (HDD and LZO).

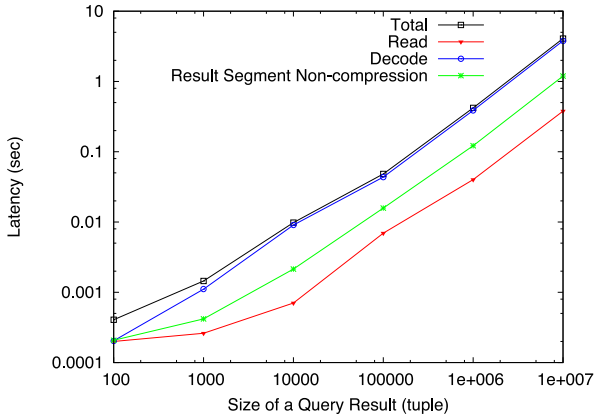

図 14 問合せ結果数に対する問合せ処理時間（SSD と LZMA）
Fig. 14 Size of a query result vs. latency (SSD and LZMA).

くなるという実験結果に一致する．

ストレージとしてソリッドステートドライブ，圧縮方式として LZMA を用いた場合の問合せ処理時間を**図 14** に示す．横軸が問合せ結果数，縦軸が問合せ処理時間である．図 14 では，「結果セグメント圧縮」の総処理時間（total），ディスクからの読み出し時間（read），圧縮データの展開時間（decode），「結果セグメント非圧縮」の総処理時間（result segment non-compression）の 4 つを描画している．図 14 から分かるように，問合せ結果数によらず，つねに「結果セグメント非圧縮」の処理時間が小さかった．また，ソリッドステートドライブと LZMA の実測データを式 (4) に代入すると

$$5.97 \times 10^6 < 15.3 \times 10^6$$

が得られ，式 (4) は成り立たない．すなわち，「結果セグメント圧縮」が「結果セグメント非圧縮」より処理速度が小さくなるという実験結果に一致する．

5.6 複数の同時問合せに関する議論

ここまでの評価は，検索クエリが 1 つだけ発生するという前提で評価を行ってきた．しかしながら，実際のシステムでは同時に多数の検索クエリが発生することも起こりうると考えられる．

そこで，複数の異なる検索クエリが同時に発生した場合の問合せ処理時間の評価を行った．評価では 5.1 節で述べた詳細評価用計算機を用いた．並列処理効果を確認するために，コアを 1 つ用いた場合と，コアを 2 つ用いた場合との比較評価を行った．ストレージにはソリッドステートドライブ，圧縮アルゴリズムには LZO と LZMA を用いた．評価では，まず，検索クエリを複数発生させ，それぞれの検索クエリごとにスレッドを生成する．各検索クエリは，範囲が重複せず，範囲のサイズが 10 万件である．次に，各スレッドはすべてのスレッドが生成されるまで待機する．すべてのスレッドが生成されると，各スレッドは問合せ処理を開始する．このとき，スレッドが処理を開始してからすべてのスレッドの処理が完了するまでの時間の計測を行った．

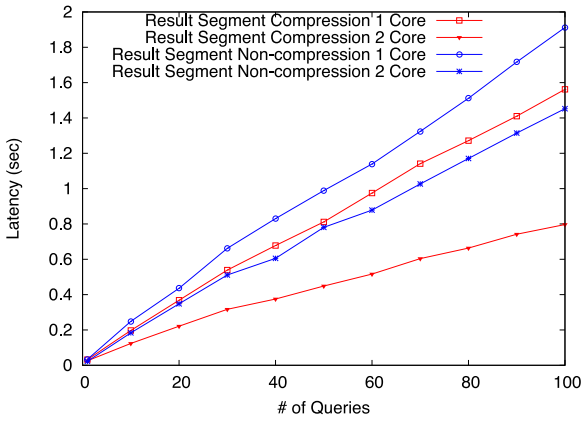

図 15 検索クエリ数に対する問合せ処理時間（LZO）
Fig. 15 # of queries vs. latency (LZO).

表 2 検索クエリ数が 100 の場合の問合せ処理時間
Table 2 Latency when # of queries is 100.

	1 コア	2 コア	改善率
結果セグメント圧縮方式（LZO）	1.56 s	0.797 s	1.96 倍
結果セグメント圧縮方式（LZMA）	4.60 s	2.68 s	1.72 倍
結果セグメント非圧縮方式	1.91 s	1.45 s	1.31 倍

図 15 に LZO を用いた場合の問合せ数に対する問合せ処理時間を示す．横軸が検索クエリ数，縦軸がすべての問合せ処理が完了するまでの時間である．LZO を用いた「結果セグメント圧縮」と「結果セグメント非圧縮」それぞれに対して，1 コアの場合と 2 コアの場合を描画している．

図 15 から分かるように，「結果セグメント圧縮」が「結果セグメント非圧縮」よりもつねに高い性能を出している．**表 2** は問合せ数が 100 のときの処理時間を抜き出したものである．表 2 から分かるように，コア数が 2 になったときの性能向上の度合いも「結果セグメント圧縮」の方が大きい．複数の検索クエリが同時発生した場合には，「結果

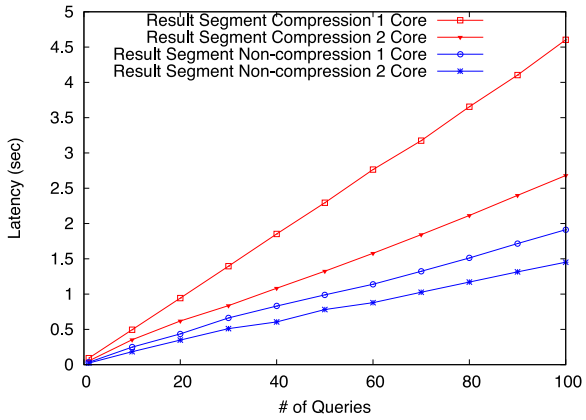

図 16 検索クエリ数に対する問合せ処理時間（LZMA）
Fig. 16 # of queries vs. latency (LZMA).

セグメント非圧縮」よりも「結果セグメント圧縮」の方がマルチコアによる並列化の効果が大きく現れている．

次に，**図 16** に LZMA を用いた場合の問合せ数に対する問合せ処理時間を示す．横軸が検索クエリ数，縦軸がすべての問合せ処理が完了するまでの時間である．LZMA を用いた「結果セグメント圧縮」と「結果セグメント非圧縮」それぞれに対して，1 コアの場合と 2 コアの場合を描画している．図 16 から分かるように，「結果セグメント圧縮」は「結果セグメント非圧縮」よりもつねに低い性能を出している．

以上より，複数の問合せが同時に発行される場合においては，「結果セグメント圧縮」はつねに有効であるわけではないことが分かる．LZO を用いた場合が高性能化する理由は，展開コストが LZMA の場合に比べて低いからである．すなわち，「結果セグメント圧縮」と「結果セグメント非圧縮」のどちらが高性能になるかは，圧縮アルゴリズムの性能に依存すると考えられる．

一方，図 16 における提案手法について，コア数が 2 の場合に性能が 1.72 倍（表 2）程度向上している点に注目されたい．この理由はコア数が増えたために並列度が向上したからだと考えられる．すなわち，コア数が増大すれば，「結果セグメント圧縮」は「結果セグメント非圧縮」よりも性能が向上すると考えられる．

以上をまとめると，複数の問合せが同時に発行される場合においては，「結果セグメント圧縮」はつねに最良であるわけではない．「結果セグメント圧縮」の性能は圧縮アルゴリズムとコア数に依存する．実験結果により，「結果セグメント圧縮」が有利である状況は，圧縮アルゴリズムがより高速であり，また，コア数がより多い場合であると考えられる．

5.7 問合せ範囲の重複（オーバラップ）に関する議論

問合せ結果が互いに重複している場合に，セグメントが小さくなったりセグメント数が増えたりすることで問合せ処理時間に影響を与える可能性がある．そこで，問合せの重複サイズや重複数が変わった場合の問合せ処理時間の評価を行った．重複サイズとは，複数の問合せが重複した際に，共有される結果セグメントのサイズである．重複数とは，複数の問合せが重複した際に，共有される結果セグメントの数である．評価では，5.1 節で述べた詳細評価用計算機を用いた．詳細評価用計算機では，ストレージとしてハードディスクを用いた．

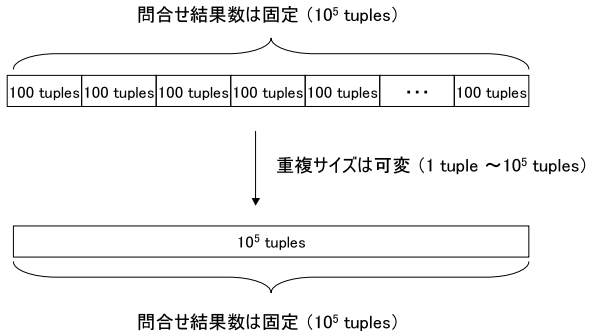

図 17 重複サイズに対する問合せ処理時間の評価方法
Fig. 17 Evaluation method of overlap size vs. latency.

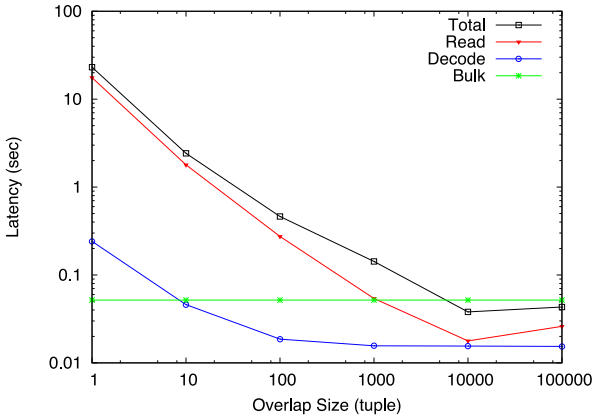

図 18 重複サイズに対する問合せ処理時間
Fig. 18 Overlap size vs. latency.

5.7.1 重複サイズが変わる場合

まず，重複サイズが変わる場合の評価を行った．評価方法を**図 17** に示す．問合せ結果数は 10 万タプルに固定した．重複サイズは 100 タプルから 10 万タプルまで変動させた．すなわち，問合せ結果である 10 万タプルの中で，複数の問合せが重複することにより，複数のセグメントが生じる状況を作り出した．

図 18 に重複サイズに対する問合せ処理時間を示す．横軸が重複サイズ，縦軸が問合せ処理に要した時間である．圧縮方式には LZO を用いた．図 18 では，「結果セグメント圧縮」の総処理時間（total），ハードディスクからの読み込みに要した時間（read），圧縮データの展開に要した時間（decode）に加えて，10 万タプルをセグメンテーションも圧縮もせずに読み出した時間（bulk）を描画している．

図 18 から，重複サイズが小さい場合には，「結果セグメ

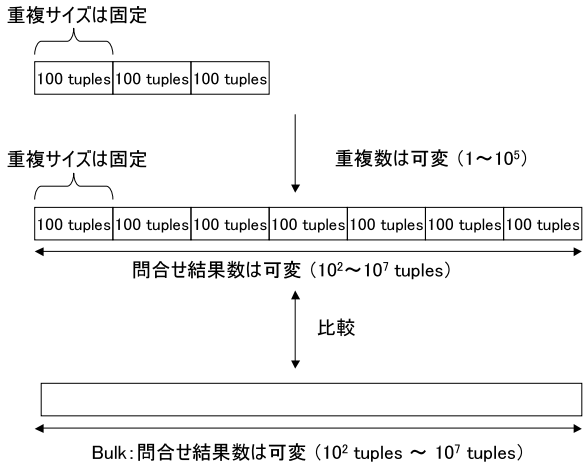

図 19 重複数に対する問合せ処理時間の評価方法
Fig. 19 Evaluation method of # of overlaps vs. latency.

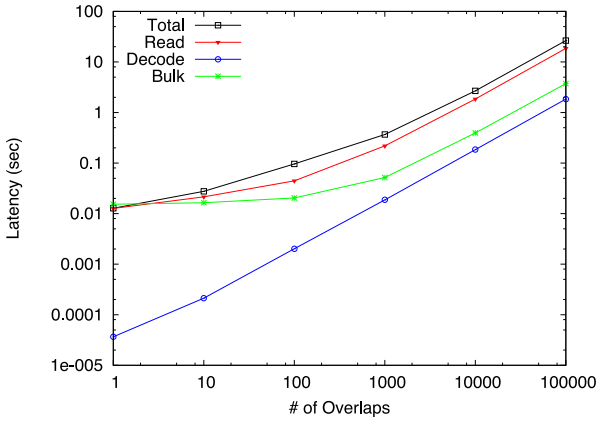

図 20 重複数に対する問合せ処理時間
Fig. 20 # of overlaps vs. latency.

ント圧縮」の性能が低いことが分かる．この理由は，重複サイズが小さい場合にはセグメントが多く発生するため，ディスクからの読み出しごとにかかるシーク時間や回転待ち時間の重畳，圧縮性能の低下によるオーバヘッドが原因であると考えられる．また，問合せ内の重複サイズが大きい場合には圧縮の効果が発揮されるため，圧縮せずに読み出したものよりも速く読み出すことができる．

重複サイズが 1 万タプル（1.E+04）の問合せ処理は，重複しないもの（1E+05）よりも処理時間が短い．この理由は，重複サイズが 1 万タプルのときでは，10 万タプルを 1 万タプルずつ読み出す際に，最初の 1 万タプルを読み込んだ時点で残りの 9 万タプル分をハードディスクが先読みを行ってディスクキャッシュに読み込むことで，2 回目以降の読み出しの時間が高速になるからだと考えられる．一方，重複しないもの（1E+05）では，ディスクキャッシュが利用されないため，性能が劣化すると考えられる．

5.7.2 重複数が変わる場合

次に，重複数が変化したときの問合せ処理時間の評価を行った．評価方法を**図 19** に示す．各セグメントのサイズは 100 タプルの固定値であり，問合せサイズが増加するとともに重複数も増加する．重複数が 1 のときは 100 タプルの問合せと同等となる．

図 20 に重複数に対する問合せ処理時間を示す．横軸が重複数，縦軸が問合せ処理に要した時間である．圧縮方式には LZO を用いた．図 20 では，「結果セグメント圧縮」の総処理時間（total），ハードディスクからの読み込みに要した時間（read），圧縮の展開に要した時間（decode）に加えて，同じ問合せ結果数のデータをセグメンテーションも圧縮もせずに読み出したバルクリードの時間（bulk）を描画している．

図 20 から分かるように，重複数が少ない場合には「結果セグメント圧縮」はバルクリードと同等の処理時間となる．また，重複数が増加するに従い，「結果セグメント圧縮」の総処理時間（total）はバルクリード（bulk）よりも遅くなる．特に，ディスクからの読み込み時間（read）においては，圧縮されているので読み込みデータサイズはバルクリードに比べて小さいにもかかわらず，読み込み時間がバルクリード（bulk）よりも大きくなる．これはディスクへのアクセス回数が増加するに従って，シーク時間や回転待ち時間のオーバヘッドが重畳されることに起因すると考えられる．

6. まとめと今後の課題

本論文の目的は，センサデータベースマネージャにおけるデータサイズの削減と問合せ処理高速化を実現することであった．本論文は問合せの自由度を犠牲にすることで性能を大幅に向上させる事前問合せ登録方式 [10], [19] を基礎とし，それに圧縮機能を導入するアプローチを提案した．同アプローチに従い，「結果セグメント圧縮」と「結果セグメント非圧縮」を示した．これらの手法はデータ領域を問合せ結果セグメントと問合せ結果以外のセグメントの 2 種類に分割する．「結果セグメント非圧縮」は問合せ結果データセグメントは圧縮せず，問合せ結果以外のセグメントを圧縮する．「結果セグメント圧縮」は両方のセグメントを圧縮する．

TIVA データベースマネージャに提案手法を含む 4 手法を実装し，データサイズと問合せ処理性能に関する実験評価を行った．実験の結果，実運用を想定した環境では，提案手法である「結果セグメント圧縮」が，データサイズの削減と問合せ処理高速化を同時に最適化するという結論を得た．

今後の課題は 3 つある．1 つ目の課題は，「結果セグメント圧縮」の改良である．実験において圧縮性能の高いアルゴリズムほど展開処理時間が長くなることが示された．展開処理を並列化することで高圧縮性能と高速展開処理の実現に取り組み，「結果セグメント圧縮」の問合せ処理性能を改善する予定である．2 つ目の課題は，提案手法の対象

範囲をデータベースマネージャからリレーショナルデータベースシステムへ拡張することである．3つ目の課題は，再編成コストの改善である．データ再編成のコストは，新しいセンサデータの到着時と，新しい問合せの発生時に生じる．データ再編成時においても，データサイズと処理時間のトレードオフが発生するため，データサイズの縮小化と再編成処理の高速化を両立させる仕組みが求められる．

参考文献

[1] Abadi, D., Madden, S. and Ferreira, M.: Integrating Compression and Execution in Column-Oriented Database Systems, *Proc. 35th ACM SIGMOD International Conference on Management of Data* (*SIGMOD'06*), Chicago, Illinois, pp.671–682 (2006).

[2] Agrawal, S., Chaudhuri, S. and Narasayya, V.: Automated Selection of Materialized Views and Indexes in SQL Databases, *Proc. 26th International Conference on Very Large Data Bases* (*VLDB'00*), Cairo, Egypt, pp.496–505 (2000).

[3] Berkeley DB, available from ⟨http://www.oracle.com/technetwork/database/berkeleydb/overview/⟩.

[4] Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R.E.: Bigtable: A Distributed Storage System for Structured Data, *Proc. 7th USENIX Symposium on Operating Systems Design and Implementation* (*OSDI'06*), Seattle, Washington, pp.205–218 (2006).

[5] Chaudhuri, S., Krishnamurthy, R., Potamianos, S. and Shim, K.: Optimizing Queries with Materialized Views, *Proc. 11th International Conference on Data Engineering* (*ICDE'95*), Taipei, Taiwan, pp.190–200 (1995).

[6] Chen, C.M.M. and Roussopoulos, N.: The Implementation and Performance Evaluation of the ADMS Query Optimizer: Integrating Query Result Caching and Matching, *Proc. International Conference on Extending Database Technology* (*EDBT'94*), Cambridge, United Kingdom, pp.323–336 (1994).

[7] Gandhi, S., Nath, S., Suri, S. and Liu, J.: GAMPS: Compressing Multi Sensor Data by Grouping and Amplitude Scaling, *Proc. 35th SIGMOD International Conference on Management of Data* (*SIGMOD'09*), Providence, Rhode Island, pp.771–784 (2009).

[8] Graefe, G. and Shapiro, L.D.: Data Compression and Database Performance, *ACM/IEEE-CS Symposium on Applied Computing* (*SAC'91*), Kansas City, Missouri, pp.22–27 (1991).

[9] Ivanova, M.G., Nes, N.J., Goncalves, R.A. and Kersten, M.L.: MonetDB/SQL Meets SkyServer: The Challenges of a Scientific Database, *Proc. 19th International Conference on Scientific and Statistical Database Management* (*SSDBM'07*), p.13, IEEE Computer Society, Washington, DC, USA (2007).

[10] Kallman, R., Kimura, H., Natkins, J., Pavlo, A., Rasin, A., Zdonik, S., Jones, E.P.C., Madden, S., Stonebraker, M., Zhang, Y., Hugg, J. and Abadi, D.J.: H-store: A High-Performance, Distributed Main Memory Transaction Processing System, *Proc. VLDB Endowment*, Vol.1, No.2, pp.1496–1499 (2008).

[11] Kurata, N., Suzuki, M., Saruwatari, S. and Morikawa, H.: Actual Application of Ubiquitous Structural Monitoring System using Wireless Sensor Networks, *Proc. 14th World Conference on Earthquake Engineering* (*14WCEE*), Beijing, China, pp.1–9 (2008).

[12] Liang, C.J.M., Liu, J., Luo, L., Terzis, A. and Zhao, F.: RACNet: A High-Fidelity Data Center Sensing Network, *Proc. 7th ACM Conference on Embedded Networked Sensor Systems* (*SenSys'09*), Berkeley, California, pp.15–28 (2009).

[13] Okamura, D.: Partially Decodable Compression with Static PPM, *Proc. Data Compression Conference* (*DCC'05*), Snowbird, Utah, p.471 (2005).

[14] Ramakrishnan, R. and Gehrke, J.: *Database Management Systems*, 3rd edition, McGraw-Hill Higher Education (2002).

[15] Reeves, G., Liu, J., Nath, S. and Zhao, F.: Managing Massive Time Series Streams with Multi-Scale Compressed Trickles, *Proc. VLDB Endowment*, Vol.2, No.1, pp.97–108 (2009).

[16] Roth, M.A. and Horn, S.J.V.: Database Compression, *ACM SIGMOD Record*, Vol.22, No.3, pp.31–39 (1993).

[17] Saruwatari, S., Kashima, T., Minami, M., Morikawa, H. and Aoyama, T.: PAVENET: A Hardware and Software Framework for Wireless Sensor Networks, *Transaction of the Society of Instrument and Control Engineers*, Vol.E-S-1, No.1, pp.74–84 (2005).

[18] Saruwatari, S., Suzuki, M. and Morikawa, H.: A Compact Hard Real-time Operating System for Wireless Sensor Networks, *Proc. 6th International Conference on Networked Sensing Systems* (*INSS'09*), Pittsburgh, Pennsylvania (2009).

[19] Stonebraker, M., Madden, S., Abadi, D.J., Harizopoulos, S., Hachem, N. and Helland, P.: The End of an Architectural Era: (It's Time for a Complete Rewrite), *Proc. 33rd International Conference on Very Large Data Bases* (*VLDB'07*), Vienna, Austria, pp.1150–1160 (2007).

[20] Stonebraker, M., Abadi, D.J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., O'Neil, P., Rasin, A., Tran, N. and Zdonik, S.: C-Store: A Column-oriented DBMS, *Proc. 31st International Conference on Very Large Data Bases* (*VLDB'05*), Trondheim, Norway, pp.553–564 (2005).

[21] Tan, K.L., Goh, S.T. and Ooi, B.C.: Cache-on-demand: Recycling with Certainty, *Proc. 17th International Conference on Data Engineering* (*ICDE'01*), Heidelberg, Germany, pp.633–640 (2001).

[22] TIVA, available from ⟨http://www.ipa.go.jp/jinzai/mitou/2009/2009_2/hontai/gaiyou/fa-2.html⟩.

[23] Tokyo Cabinet, available from ⟨http://1978th.net/tokyocabinet⟩.

[24] VoltDB Technical Overview, Next-Generation Open-Source SQL Database with ACID for Fast-Scaling OLTP Applications, VoltDB Cooporation White Paper (2010).

[25] 首藤一幸：スケールアウトの技術，情報処理，Vol.50, No.11, pp.1080–1085 (2009).

猿渡 俊介 （正会員）

2007 年東京大学大学院博士課程修了．2003～2004 年 IPA 未踏ソフトウェア創造事業，2006～2008 年日本学術振興会学振特別研究員，2007～2008 年イリノイ大学客員研究員，現在，東京大学先端科学技術研究センター助教．専門はワイヤレスネットワーク，センサネットワーク，システムソフトウェア等．2009 年電子情報通信学会論文賞．2010 年情報処理学会山下記念研究賞．電子情報通信学会，IEEE，ACM 各会員．

高木 潤一郎

2009 年東京大学工学部電子情報工学科卒業．2011 年同大学大学院工学系研究科電気系工学専攻融合情報学コース修了．2009～2010 年 IPA 未踏人材発掘・育成事業．現在，ソニー株式会社．専門はセンサデータベース等．

川島 英之 （正会員）

1999 年慶應義塾大学理工学部電気工学科卒業．2005 年同大学大学院理工学研究科開放環境科学専攻後期博士課程修了．同年慶應義塾大学理工学部助手．2007 年筑波大学大学院システム情報工学研究科講師，ならびに計算科学研究センター講師．2011 年筑波大学システム情報系講師．博士（工学）．センタデータ管理に関する研究に従事．日本データベース学会，ACM，IEEE 各会員．

倉田 成人

1986 年東京大学大学院工学系研究科建築学専門課程（地震研究所）修士課程修了．1986 年鹿島建設（株）入社．地盤と構造物の相互作用，可変剛性・減衰システム，セミアクティブ制震システム，ユビキタス・センサネットワークによる構造モニタリング，建築空間のエネルギーモニタリングの研究に従事．専門は地震工学，構造工学，センサネットワーク等．現在，同社技術研究所上席研究員．博士（工学）．IEEE，日本建築学会，計測自動制御学会各会員．

森川 博之 （正会員）

1987 年東京大学工学部電子工学科卒業．1992 年同大学大学院博士課程修了．1997～1998 年コロンビア大学客員研究員．2002～2006 年情報通信研究機構モバイルネットワークグループリーダ兼務．現在，東京大学先端科学技術研究センター教授．専門はユビキタスネットワーク，モバイルコンピューティング，ワイヤレスネットワーク，フォトニックインターネット等．本会論文賞，電子情報通信学会論文賞（3 回），志田林三郎賞，情報通信月間推進協議会情報通信功績賞，ドコモモバイルサイエンス賞等受賞．電子情報通信学会フェロー．IEEE，ACM，ISOC，映像情報メディア学会各会員．